On Sale - WEB Часть II

Оглавление

**Управление пользователями 2**

**API для рейтингов 9**

**Улучшим** [**seeder**](#_clbked6lkjhd) **20**

**Публикация тостов 26**

**API для регистрации пользователя 27**

**API для восстановления пароля 30**

**API для изменения пользователя 31**

**API для смены пароля 32**

**API создавать заказы 34**

**Улучшение просмотра комментариев 38**

**Управлять заказами 40**

**API для получения истории заказов 53**

**API для изменения заказа (отмена заказа) 54**

**Конец 55**

# 

# Управление пользователями

1. Добавьте этот метод в MVC **AccountController**:

[Authorize(Roles = "Admin")]

public async Task<IActionResult> Index()

{

return View(await \_context.Users

.Include(u => u.City)

.ToListAsync());

}

1. Добавить в представление **Index**:

@model IEnumerable<OnSale.Web.Data.Entities.User>

@{

ViewData["Title"] = "Index";

}

<link rel="stylesheet" href="https://cdn.datatables.net/1.10.19/css/jquery.dataTables.min.css" />

<br />

<p>

<a asp-action="Create" class="btn btn-primary"><i class="glyphicon glyphicon-plus"></i> New Admin</a>

</p>

<div class="row">

<div class="col-md-12">

<div class="panel panel-default">

<div class="panel-heading">

<h3 class="panel-title">Users</h3>

</div>

<div class="panel-body">

<table class="table table-hover table-responsive table-striped" id="MyTable">

<thead>

<tr>

<th>

@Html.DisplayNameFor(model => model.FullName)

</th>

<th>

@Html.DisplayNameFor(model => model.UserType)

</th>

<th>

@Html.DisplayNameFor(model => model.Email)

</th>

<th>

@Html.DisplayNameFor(model => model.City.Name)

</th>

<th>

@Html.DisplayNameFor(model => model.Address)

</th>

<th>

@Html.DisplayNameFor(model => model.PhoneNumber)

</th>

<th>

@Html.DisplayNameFor(model => model.ImageFullPath)

</th>

</tr>

</thead>

<tbody>

@foreach (var item in Model)

{

<tr>

<td>

@Html.DisplayFor(modelItem => item.FullName)

</td>

<td>

@Html.DisplayFor(modelItem => item.UserType)

</td>

<td>

@Html.DisplayFor(modelItem => item.Email)

</td>

<td>

@Html.DisplayFor(modelItem => item.City.Name)

</td>

<td>

@Html.DisplayFor(modelItem => item.Address)

</td>

<td>

@Html.DisplayFor(modelItem => item.PhoneNumber)

</td>

<td>

<img src="@item.ImageFullPath" style="width:80px;height:80px;border-radius:50%" />

</td>

</tr>

}

</tbody>

</table>

</div>

</div>

</div>

</div>

@section Scripts {

@{await Html.RenderPartialAsync("\_ValidationScriptsPartial");}

<script src="//cdn.datatables.net/1.10.19/js/jquery.dataTables.min.js"></script>

<script src="/js/deleteDialog.js"></script>

<script type="text/javascript">

$(document).ready(function () {

$('#MyTable').DataTable();

});

</script>

}

1. Добавьте эти методы в MVC **AccountController**:

[Authorize(Roles = "Admin")]

[HttpGet]

public IActionResult Create()

{

AddUserViewModel model = new AddUserViewModel

{

Countries = \_combosHelper.GetComboCountries(),

Departments = \_combosHelper.GetComboDepartments(0),

Cities = \_combosHelper.GetComboCities(0),

};

return View(model);

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Create(AddUserViewModel model)

{

if (ModelState.IsValid)

{

Guid imageId = Guid.Empty;

if (model.ImageFile != null)

{

imageId = await \_blobHelper.UploadBlobAsync(model.ImageFile, "users");

}

User user = await \_userHelper.AddUserAsync(model, imageId, UserType.Admin);

if (user == null)

{

ModelState.AddModelError(string.Empty, "This email is already used.");

model.Countries = \_combosHelper.GetComboCountries();

model.Departments = \_combosHelper.GetComboDepartments(model.CountryId);

model.Cities = \_combosHelper.GetComboCities(model.DepartmentId);

return View(model);

}

string myToken = await \_userHelper.GenerateEmailConfirmationTokenAsync(user);

string tokenLink = Url.Action("ConfirmEmail", "Account", new

{

userid = user.Id,

token = myToken

}, protocol: HttpContext.Request.Scheme);

Response response = \_mailHelper.SendMail(model.Username, "Email confirmation", $"<h1>Email Confirmation</h1>" +

$"To allow the user, " +

$"plase click in this link:</br></br><a href = \"{tokenLink}\">Confirm Email</a>");

if (response.IsSuccess)

{

ViewBag.Message = "The instructions to allow your user has been sent to email.";

return View(model);

}

ModelState.AddModelError(string.Empty, response.Message);

}

model.Countries = \_combosHelper.GetComboCountries();

model.Departments = \_combosHelper.GetComboDepartments(model.CountryId);

model.Cities = \_combosHelper.GetComboCities(model.DepartmentId);

return View(model);

}

1. Добавить представление **Create**:

@model OnSale.Web.Models.AddUserViewModel

@{

ViewData["Title"] = "Register";

}

<h2>Add</h2>

<h4>Admin</h4>

<hr />

<div class="row">

<div class="col-md-4">

<form asp-action="Create" enctype="multipart/form-data">

<div asp-validation-summary="ModelOnly" class="text-danger"></div>

<div class="form-group">

<label asp-for="Username" class="control-label"></label>

<input asp-for="Username" class="form-control" />

<span asp-validation-for="Username" class="text-danger"></span>

</div>

<partial name="\_User" />

<div class="form-group">

<label asp-for="Password" class="control-label"></label>

<input asp-for="Password" class="form-control" />

<span asp-validation-for="Password" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="PasswordConfirm" class="control-label"></label>

<input asp-for="PasswordConfirm" class="form-control" />

<span asp-validation-for="PasswordConfirm" class="text-danger"></span>

</div>

<div class="form-group">

<input type="submit" value="Register" class="btn btn-primary" />

</div>

</form>

</div>

</div>

<div class="text-success">

<p>

@ViewBag.Message

</p>

</div>

@section Scripts {

@{await Html.RenderPartialAsync("\_ValidationScriptsPartial");}

<script type="text/javascript">

$(document).ready(function () {

$("#CountryId").change(function () {

$("#DepartmentId").empty();

$("#DepartmentId").append('<option value="0">[Select a department...]</option>');

$("#CityId").empty();

$("#CityId").append('<option value="0">[Select a city...]</option>');

$.ajax({

type: 'POST',

url: '@Url.Action("GetDepartments")',

dataType: 'json',

data: { countryId: $("#CountryId").val() },

success: function (subcategories) {

$.each(subcategories, function (i, department) {

debugger;

$("#DepartmentId").append('<option value="'

+ department.id + '">'

+ department.name + '</option>');

});

},

error: function (ex) {

alert('Failed to retrieve departments.' + ex);

}

});

return false;

})

$("#DepartmentId").change(function () {

$("#CityId").empty();

$("#CityId").append('<option value="0">[Select a city...]</option>');

$.ajax({

type: 'POST',

url: '@Url.Action("GetCities")',

dataType: 'json',

data: { departmentId: $("#DepartmentId").val() },

success: function (cities) {

$.each(cities, function (i, city) {

debugger;

$("#CityId").append('<option value="'

+ city.id + '">'

+ city.name + '</option>');

});

},

error: function (ex) {

alert('Failed to retrieve cities.' + ex);

}

});

return false;

})

});

</script>

}

1. Тестируем.

# 

# API для рейтингов

1

1

1

1

1

1

1

1

1

1

N

N

N

N

N

N

N

N

N

1. Давайте добавим объект **Qualification** внутри **Web.Data.Entities:**

public class Qualification

{

public int Id { get; set; }

[DisplayFormat(DataFormatString = "{0:yyyy/MM/dd hh:mm}")]

public DateTime Date { get; set; }

[Display(Name = "Date")]

[DisplayFormat(DataFormatString = "{0:yyyy/MM/dd hh:mm}")]

public DateTime DateLocal => Date.ToLocalTime();

[JsonIgnore]

public Product Product { get; set; }

public User User { get; set; }

[DisplayFormat(DataFormatString = "{0:N2}")]

public float Score { get; set; }

[DataType(DataType.MultilineText)]

public string Remarks { get; set; }

}

1. Мы перемещаем сущности **Product**, **OrderDetail** в **Web.Data.Entities**.
2. Мы добавляем эти атрибуты в **Product**:

public ICollection<Qualification> Qualifications { get; set; }

[DisplayName("Product Qualifications")]

public int ProductQualifications => Qualifications == null ? 0 : Qualifications.Count;

[DisplayFormat(DataFormatString = "{0:N2}")]

public float Qualification => Qualifications == null || Qualifications.Count == 0 ? 0 : Qualifications.Average(q => q.Score);

1. Внутри **Common.Responses** добавляем **QualificationResponse**:

public class QualificationResponse

{

public int Id { get; set; }

public DateTime Date { get; set; }

public float Score { get; set; }

public string Remarks { get; set; }

}

1. Внутри **Common.Responses** добавляем **ProductResponse**:

public class ProductResponse

{

public int Id { get; set; }

public string Name { get; set; }

public string Description { get; set; }

public decimal Price { get; set; }

public bool IsActive { get; set; }

public bool IsStarred { get; set; }

public Category Category { get; set; }

public ICollection<ProductImage> ProductImages { get; set; }

public int ProductImagesNumber => ProductImages == null ? 0 : ProductImages.Count;

public string ImageFullPath => ProductImages == null || ProductImages.Count == 0

? $"https://onsaleprepweb.azurewebsites.net/images/noimage.png"

: ProductImages.FirstOrDefault().ImageFullPath;

public ICollection<QualificationResponse> Qualifications { get; set; }

public int ProductQualifications => Qualifications == null ? 0 : Qualifications.Count;

public float Qualification => Qualifications == null || Qualifications.Count == 0 ? 0 : Qualifications.Average(q => q.Score);

}

1. Мы исправили "беспорядок", который сделали с проектом.
2. Мы добавляем новую сущность в **DataContext**:

public DbSet<Qualification> Qualifications { get; set; }

1. Запускаем команды для добавления новой миграции и обновления базы данных.
2. Мы добавили запрос **QualificationRequest**:

public class QualificationRequest

{

[Required]

public int ProductId { get; set; }

[Range(0, 5)]

[Required]

public float Score { get; set; }

public string Remarks { get; set; }

}

1. Создаем **API**-контроллер для квалификаций, **QualificationsController**:

[ApiController]

[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]

[Route("api/[controller]")]

public class QualificationsController : ControllerBase

{

private readonly DataContext \_context;

private readonly IUserHelper \_userHelper;

public QualificationsController(DataContext context, IUserHelper userHelper)

{

\_context = context;

\_userHelper = userHelper;

}

[HttpPost]

public async Task<IActionResult> PostQualification([FromBody] QualificationRequest request)

{

if (!ModelState.IsValid)

{

return BadRequest();

}

string email = User.Claims.FirstOrDefault(c => c.Type == ClaimTypes.NameIdentifier).Value;

User user = await \_userHelper.GetUserAsync(email);

if (user == null)

{

return NotFound("Error001");

}

Product product = await \_context.Products

.Include(p => p.Qualifications)

.FirstOrDefaultAsync(p => p.Id == request.ProductId);

if (product == null)

{

return NotFound("Error002");

}

if (product.Qualifications == null)

{

product.Qualifications = new List<Qualification>();

}

product.Qualifications.Add(new Qualification

{

Date = DateTime.UtcNow,

Product = product,

Remarks = request.Remarks,

Score = request.Score,

User = user

});

\_context.Products.Update(product);

await \_context.SaveChangesAsync();

return Ok(product);

}

}

1. Modificamos el API de productos:

[HttpGet]

public async Task<IActionResult> GetProducts()

{

List<Product> products = await \_context.Products

.Include(p => p.Category)

.Include(p => p.ProductImages)

.Include(p => p.Qualifications)

.Where(p => p.IsActive)

.ToListAsync();

return Ok(products);

}

1. Probamos.
2. Solucionamos el problema de seguridad que tenemos con este Endpoint:

[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]

[HttpPost]

public async Task<IActionResult> GetUser()

{

if (!ModelState.IsValid)

{

return BadRequest();

}

string email = User.Claims.FirstOrDefault(c => c.Type == ClaimTypes.NameIdentifier).Value;

User user = await \_userHelper.GetUserAsync(email);

if (user == null)

{

return NotFound("Error001");

}

return Ok(user);

}

1. Modificamos el **ProductsController** MVC:

public async Task<IActionResult> Index()

{

return View(await \_context.Products

.Include(p => p.Category)

.Include(p => p.ProductImages)

.Include(p => p.Qualifications)

.ToListAsync());

}

…

public async Task<IActionResult> Details(int? id)

{

if (id == null)

{

return NotFound();

}

Product product = await \_context.Products

.Include(c => c.Category)

.Include(c => c.ProductImages)

.Include(c => c.Qualifications)

.ThenInclude(q => q.User)

.FirstOrDefaultAsync(m => m.Id == id);

if (product == null)

{

return NotFound();

}

return View(product);

}

1. Modificamos la vista **Index**:

...

<th>

@Html.DisplayNameFor(model => model.IsStarred)

</th>

<th>

@Html.DisplayNameFor(model => model.Qualification)

</th>

<th>

Category

</th>

<th>

@Html.DisplayNameFor(model => model.ProductImagesNumber)

</th>

<th>

@Html.DisplayNameFor(model => model.ProductQualifications)

</th>

<th width="120px"></th>

…

<td>

@Html.DisplayFor(modelItem => item.Qualification)

</td>

<td>

@Html.DisplayFor(modelItem => item.Category.Name)

</td>

<td>

@Html.DisplayFor(modelItem => item.ProductImagesNumber)

</td>

<td>

@Html.DisplayFor(modelItem => item.ProductQualifications)

</td>

...

1. Modificamos la vista **Details**:

@model OnSale.Web.Data.Entities.Product

@{

ViewData["Title"] = "Details";

}

<link rel="stylesheet" href="https://cdn.datatables.net/1.10.19/css/jquery.dataTables.min.css" />

<h2>Details</h2>

<div>

<h4>Product</h4>

<hr />

<dl class="dl-horizontal">

<dt>

@Html.DisplayNameFor(model => model.Name)

</dt>

<dd>

@Html.DisplayFor(model => model.Name)

</dd>

<dt>

@Html.DisplayNameFor(model => model.Description)

</dt>

<dd>

@Html.DisplayFor(model => model.Description)

</dd>

<dt>

Category

</dt>

<dd>

@Html.DisplayFor(model => model.Category.Name)

</dd>

<dt>

@Html.DisplayNameFor(model => model.Price)

</dt>

<dd>

@Html.DisplayFor(model => model.Price)

</dd>

<dt>

@Html.DisplayNameFor(model => model.IsActive)

</dt>

<dd>

@Html.DisplayFor(model => model.IsActive)

</dd>

<dt>

@Html.DisplayNameFor(model => model.IsStarred)

</dt>

<dd>

@Html.DisplayFor(model => model.IsStarred)

</dd>

<dt>

@Html.DisplayNameFor(model => model.Qualification)

</dt>

<dd>

@Html.DisplayFor(model => model.Qualification)

</dd>

<dt>

@Html.DisplayNameFor(model => model.ProductImagesNumber)

</dt>

<dd>

@Html.DisplayFor(model => model.ProductImagesNumber)

</dd>

<dt>

@Html.DisplayNameFor(model => model.ProductQualifications)

</dt>

<dd>

@Html.DisplayFor(model => model.ProductQualifications)

</dd>

</dl>

</div>

<div>

<a asp-action="AddImage" asp-route-id="@Model.Id" class="btn btn-primary"><i class="glyphicon glyphicon-plus"></i> Image</a>

<a asp-action="Edit" asp-route-id="@Model.Id" class="btn btn-warning">Edit</a>

<a asp-action="Index" class="btn btn-success">Back to List</a>

</div>

<br />

<div class="row">

<div class="col-md-5">

<div class="panel panel-default">

<div class="panel-heading">

<h3 class="panel-title">Product Images</h3>

</div>

<div class="panel-body">

<table class="table table-hover table-responsive table-striped" id="MyTableImages">

<thead>

<tr>

<th>

@Html.DisplayNameFor(model => model.ProductImages.FirstOrDefault().ImageFullPath)

</th>

<th></th>

</tr>

</thead>

<tbody>

@foreach (var item in Model.ProductImages)

{

<tr>

<td>

<img src="@item.ImageFullPath" style="width:200px;height:200px;max-width: 100%; height: auto;" />

</td>

<td>

<button data-id="@item.Id" class="btn btn-danger deleteItem" data-toggle="modal" data-target="#deleteDialog"><i class="glyphicon glyphicon-trash"></i></button>

</td>

</tr>

}

</tbody>

</table>

</div>

</div>

</div>

<div class="col-md-7">

<div class="panel panel-default">

<div class="panel-heading">

<h3 class="panel-title">Qualifications</h3>

</div>

<div class="panel-body">

<table class="table table-hover table-responsive table-striped" id="MyTableQualifications">

<thead>

<tr>

<th>

@Html.DisplayNameFor(model => model.Qualifications.FirstOrDefault().DateLocal)

</th>

<th>

@Html.DisplayNameFor(model => model.Qualifications.FirstOrDefault().User.Email)

</th>

<th>

@Html.DisplayNameFor(model => model.Qualifications.FirstOrDefault().Score)

</th>

<th>

@Html.DisplayNameFor(model => model.Qualifications.FirstOrDefault().Remarks)

</th>

</tr>

</thead>

<tbody>

@foreach (var item in Model.Qualifications)

{

<tr>

<td>

@Html.DisplayFor(modelItem => item.DateLocal)

</td>

<td>

@Html.DisplayFor(modelItem => item.User.Email)

</td>

<td>

@Html.DisplayFor(modelItem => item.Score)

</td>

<td>

@Html.DisplayFor(modelItem => item.Remarks)

</td>

</tr>

}

</tbody>

</table>

</div>

</div>

</div>

</div>

<partial name="\_DeleteDialog" />

@section Scripts {

@{await Html.RenderPartialAsync("\_ValidationScriptsPartial");}

<script src="//cdn.datatables.net/1.10.19/js/jquery.dataTables.min.js"></script>

<script src="/js/deleteDialog.js"></script>

<script type="text/javascript">

$(document).ready(function () {

$('#MyTableImages').DataTable();

$('#MyTableQualifications').DataTable();

// Delete item

sc\_deleteDialog.openModal('deleteItem', true, 'btnYesDelete', '/Products/DeleteImage/', false);

});

</script>

}

1. Probamos.
2. Borramos BD y publicamos de nuevo en Azure.

# Mejorar el seeder

1. Adicionar las imágenes que el Seeder va a utilizar.
2. Modificamos el **SeedDb**:

public class SeedDb

{

private readonly DataContext \_context;

private readonly IUserHelper \_userHelper;

private readonly IBlobHelper \_blobHelper;

private readonly Random \_random;

public SeedDb(DataContext context, IUserHelper userHelper, IBlobHelper blobHelper)

{

\_context = context;

\_userHelper = userHelper;

\_blobHelper = blobHelper;

\_random = new Random();

}

public async Task SeedAsync()

{

await \_context.Database.EnsureCreatedAsync();

await CheckCountriesAsync();

await CheckRolesAsync();

await CheckUserAsync("1010", "Juan", "Zuluaga", "jzuluaga55@hotmail.com", "322 311 4620", "Calle Luna Calle Sol", UserType.Admin);

await CheckCategoriesAsync();

await CheckProductsAsync();

}

private async Task CheckProductsAsync()

{

if (!\_context.Products.Any())

{

User user = await \_userHelper.GetUserAsync("jzuluaga55@hotmail.com");

Category mascotas = await \_context.Categories.FirstOrDefaultAsync(c => c.Name == "Mascotas");

Category ropa = await \_context.Categories.FirstOrDefaultAsync(c => c.Name == "Ropa");

Category tecnologia = await \_context.Categories.FirstOrDefaultAsync(c => c.Name == "Tecnología");

string lorem = "Lorem ipsum dolor sit amet, consectetur adipiscing elit. Mauris gravida, nunc vel tristique cursus, velit nibh pulvinar enim, non pulvinar lorem leo eget felis. Proin suscipit dignissim nisl, at elementum justo laoreet sed. In tortor nibh, auctor quis est gravida, blandit elementum nulla. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Integer placerat nisi dui, id rutrum nisi viverra at. Interdum et malesuada fames ac ante ipsum primis in faucibus. Pellentesque sodales sollicitudin tempor. Fusce volutpat, purus sit amet placerat gravida, est magna gravida risus, a ultricies augue magna vel dolor. Fusce egestas venenatis velit, a ultrices purus aliquet sed. Morbi lacinia purus sit amet nisi vulputate mollis. Praesent in volutpat tortor. Etiam ac enim id ligula rutrum semper. Sed mattis erat sed condimentum congue. Vestibulum consequat tristique consectetur. Nunc in lorem in sapien vestibulum aliquet a vel leo.";

await AddProductAsync(mascotas, lorem, "Bulldog Frances", 2500000M, new string[] { "Bulldog1", "Bulldog2", "Bulldog3", "Bulldog4" }, user);

await AddProductAsync(ropa, lorem, "Buso GAP Hombre", 85000M, new string[] { "BusoGAP1", "BusoGAP2" }, user);

await AddProductAsync(tecnologia, lorem, "iPhone 11", 3500000M, new string[] { "iPhone1", "iPhone2", "iPhone3", "iPhone4", "iPhone5" }, user);

await AddProductAsync(tecnologia, lorem, "iWatch \"42", 2100000M, new string[] { "iWatch" }, user);

await AddProductAsync(ropa, lorem, "Tennis Adidas", 250000M, new string[] { "Adidas" }, user);

await AddProductAsync(mascotas, lorem, "Collie", 350000M, new string[] { "Collie1", "Collie2", "Collie3", "Collie4", "Collie5" }, user);

await AddProductAsync(tecnologia, lorem, "MacBook Pro 16\" 1TB", 12000000M, new string[] { "MacBookPro1", "MacBookPro2", "MacBookPro3", "MacBookPro4" }, user);

await AddProductAsync(ropa, lorem, "Sudadera Mujer", 95000M, new string[] { "Sudadera1", "Sudadera2", "Sudadera3", "Sudadera4", "Sudadera5" }, user);

await \_context.SaveChangesAsync();

}

}

private async Task AddProductAsync(Category category, string description, string name, decimal price, string[] images, User user)

{

Product product = new Product

{

Category = category,

Description = description,

IsActive = true,

Name = name,

Price = price,

ProductImages = new List<ProductImage>(),

Qualifications = GetRandomQualifications(description, user)

};

foreach (string image in images)

{

string path = Path.Combine(Directory.GetCurrentDirectory(), $"wwwroot\\images", $"{image}.png");

Guid imageId = await \_blobHelper.UploadBlobAsync(path, "products");

product.ProductImages.Add(new ProductImage { ImageId = imageId });

}

\_context.Products.Add(product);

}

private ICollection<Qualification> GetRandomQualifications(string description, User user)

{

List<Qualification> qualifications = new List<Qualification>();

for (int i = 0; i < 10; i++)

{

qualifications.Add(new Qualification

{

Date = DateTime.UtcNow,

Remarks = description,

Score = \_random.Next(1, 5),

User = user

});

}

return qualifications;

}

private async Task CheckCategoriesAsync()

{

if (!\_context.Categories.Any())

{

await AddCategoryAsync("Ropa");

await AddCategoryAsync("Tecnología");

await AddCategoryAsync("Mascotas");

await \_context.SaveChangesAsync();

}

}

private async Task AddCategoryAsync(string name)

{

string path = Path.Combine(Directory.GetCurrentDirectory(), $"wwwroot\\images", $"{name}.png");

Guid imageId = await \_blobHelper.UploadBlobAsync(path, "categories");

\_context.Categories.Add(new Category { Name = name, ImageId = imageId });

}

private async Task CheckRolesAsync()

{

await \_userHelper.CheckRoleAsync(UserType.Admin.ToString());

await \_userHelper.CheckRoleAsync(UserType.User.ToString());

}

private async Task<User> CheckUserAsync(

string document,

string firstName,

string lastName,

string email,

string phone,

string address,

UserType userType)

{

User user = await \_userHelper.GetUserAsync(email);

if (user == null)

{

user = new User

{

FirstName = firstName,

LastName = lastName,

Email = email,

UserName = email,

PhoneNumber = phone,

Address = address,

Document = document,

City = \_context.Cities.FirstOrDefault(),

UserType = userType

};

await \_userHelper.AddUserAsync(user, "123456");

await \_userHelper.AddUserToRoleAsync(user, userType.ToString());

string token = await \_userHelper.GenerateEmailConfirmationTokenAsync(user);

await \_userHelper.ConfirmEmailAsync(user, token);

}

return user;

}

private async Task CheckCountriesAsync()

{

if (!\_context.Countries.Any())

{

\_context.Countries.Add(new Country

{

Name = "Colombia",

Departments = new List<Department>

{

new Department

{

Name = "Antioquia",

Cities = new List<City>

{

new City { Name = "Medellín" },

new City { Name = "Envigado" },

new City { Name = "Itagüí" }

}

},

new Department

{

Name = "Bogotá",

Cities = new List<City>

{

new City { Name = "Bogotá" }

}

},

new Department

{

Name = "Valle del Cauca",

Cities = new List<City>

{

new City { Name = "Calí" },

new City { Name = "Buenaventura" },

new City { Name = "Palmira" }

}

}

}

});

\_context.Countries.Add(new Country

{

Name = "USA",

Departments = new List<Department>

{

new Department

{

Name = "California",

Cities = new List<City>

{

new City { Name = "Los Angeles" },

new City { Name = "San Diego" },

new City { Name = "San Francisco" }

}

},

new Department

{

Name = "Illinois",

Cities = new List<City>

{

new City { Name = "Chicago" },

new City { Name = "Springfield" }

}

}

}

});

await \_context.SaveChangesAsync();

}

}

}

1. Probamos.

# Colocar mensajes tipo Toast

Gracias a Hugo Rivera Dijeres que me paso el tip.

1. Instalamos el siguiente paquete:

PM> Install-Package Vereyon.Web.FlashMessage

1. Lo registramos en el **StartUp**:

services.AddFlashMessage();

1. Lo registramos en el **\_ViewImports**:

@addTagHelper \*, Vereyon.Web.FlashMessage

1. Lo inyectamos en el controlador donde queramos el mensaje, para el ejemplo en el **CategoriesController**:

IFlashMessage flashMessage

1. Se usa de la siguiente forma, ejemplo en el **Delete** de **Categories**:

try

{

\_context.Categories.Remove(category);

await \_context.SaveChangesAsync();

\_flashMessage.Confirmation("The category was deleted.");

}

catch (Exception ex)

{

\_flashMessage.Danger("The category can't be deleted because it has related records.");

}

1. En la vista donde se mostrará el toast, en este caso la Index de Categories, adicionamos lo siguiente:

<flash dismissable="true" />

1. Probamos

Tomado de: <https://github.com/Vereyon/FlashMessage>

# API para registro de usuarios

1. Adicionar la **UserRequest**:

public class UserRequest

{

[Required]

public string Document { get; set; }

[Required]

public string FirstName { get; set; }

[Required]

public string LastName { get; set; }

[Required]

public string Address { get; set; }

[Required]

public string Email { get; set; }

[Required]

public string Phone { get; set; }

[Required]

[StringLength(20, MinimumLength = 6)]

public string Password { get; set; }

[Required]

public int CityId { get; set; }

public byte[] ImageArray { get; set; }

}

1. Modificar el API **AccountController**:

private readonly IUserHelper \_userHelper;

private readonly IConfiguration \_configuration;

private readonly IBlobHelper \_blobHelper;

private readonly IMailHelper \_mailHelper;

private readonly DataContext \_context;

public AccountController(

IUserHelper userHelper,

IConfiguration configuration,

IBlobHelper blobHelper,

IMailHelper mailHelper,

DataContext context)

{

\_userHelper = userHelper;

\_configuration = configuration;

\_blobHelper = blobHelper;

\_mailHelper = mailHelper;

\_context = context;

}

...

[HttpPost]

[Route("Register")]

public async Task<IActionResult> PostUser([FromBody] UserRequest request)

{

if (!ModelState.IsValid)

{

return BadRequest(new Response

{

IsSuccess = false,

Message = "Bad request",

Result = ModelState

});

}

User user = await \_userHelper.GetUserAsync(request.Email);

if (user != null)

{

return BadRequest(new Response

{

IsSuccess = false,

Message = "Error003"

});

}

//TODO: Translate ErrorXXX literals

City city = await \_context.Cities.FindAsync(request.CityId);

if (city == null)

{

return BadRequest(new Response

{

IsSuccess = false,

Message = "Error004"

});

}

Guid imageId = Guid.Empty;

if (request.ImageArray != null)

{

imageId = await \_blobHelper.UploadBlobAsync(request.ImageArray, "users");

}

user = new User

{

Address = request.Address,

Document = request.Document,

Email = request.Email,

FirstName = request.FirstName,

LastName = request.LastName,

PhoneNumber = request.Phone,

UserName = request.Email,

ImageId = imageId,

UserType = UserType.User,

City = city

};

IdentityResult result = await \_userHelper.AddUserAsync(user, request.Password);

if (result != IdentityResult.Success)

{

return BadRequest(result.Errors.FirstOrDefault().Description);

}

User userNew = await \_userHelper.GetUserAsync(request.Email);

await \_userHelper.AddUserToRoleAsync(userNew, user.UserType.ToString());

string myToken = await \_userHelper.GenerateEmailConfirmationTokenAsync(user);

string tokenLink = Url.Action("ConfirmEmail", "Account", new

{

userid = user.Id,

token = myToken

}, protocol: HttpContext.Request.Scheme);

\_mailHelper.SendMail(request.Email, "Email Confirmation", $"<h1>Email Confirmation</h1>" +

$"To confirm your email please click on the link<p><a href = \"{tokenLink}\">Confirm Email</a></p>");

return Ok(new Response { IsSuccess = true });

}

1. Probar en Postman.

# API para recuperación de contraseña

1. Adicionar este método al API **AccountController**:

[HttpPost]

[Route("RecoverPassword")]

public async Task<IActionResult> RecoverPassword([FromBody] EmailRequest request)

{

if (!ModelState.IsValid)

{

return BadRequest(new Response

{

IsSuccess = false,

Message = "Bad request"

});

}

User user = await \_userHelper.GetUserAsync(request.Email);

if (user == null)

{

return BadRequest(new Response

{

IsSuccess = false,

Message = "Error001"

});

}

string myToken = await \_userHelper.GeneratePasswordResetTokenAsync(user);

string link = Url.Action("ResetPassword", "Account", new { token = myToken }, protocol: HttpContext.Request.Scheme);

\_mailHelper.SendMail(request.Email, "Password Recover", $"<h1>Password Recover</h1>" +

$"Click on the following link to change your password:<p>" +

$"<a href = \"{link}\">Change Password</a></p>");

return Ok(new Response { IsSuccess = true});

}

1. En el MVC **AccountController** renombrar el método **RecoverPassword** a **RecoverPasswordMVC** para evitar que entre en conflicto en el controlador API.
2. Probamos en Postman.
3. Publicamos de nuevo en Azure.

# API para modificar usuario

1. Adicionar este método al API **AccountController**:

[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]

[HttpPut]

public async Task<IActionResult> PutUser([FromBody] UserRequest request)

{

if (!ModelState.IsValid)

{

return BadRequest(ModelState);

}

string email = User.Claims.FirstOrDefault(c => c.Type == ClaimTypes.NameIdentifier).Value;

User user = await \_userHelper.GetUserAsync(email);

if (user == null)

{

return NotFound("Error001");

}

City city = await \_context.Cities.FindAsync(request.CityId);

if (city == null)

{

return BadRequest(new Response

{

IsSuccess = false,

Message = "Error004"

});

}

Guid imageId = user.ImageId;

if (request.ImageArray != null)

{

imageId = await \_blobHelper.UploadBlobAsync(request.ImageArray, "users");

}

user.FirstName = request.FirstName;

user.LastName = request.LastName;

user.Address = request.Address;

user.PhoneNumber = request.Phone;

user.Document = request.Phone;

user.City = city;

user.ImageId = imageId;

IdentityResult respose = await \_userHelper.UpdateUserAsync(user);

if (!respose.Succeeded)

{

return BadRequest(respose.Errors.FirstOrDefault().Description);

}

User updatedUser = await \_userHelper.GetUserAsync(email);

return Ok(updatedUser);

}

1. Probamos en Postman.
2. Publicamos de nuevo en Azure.

# API para cambiar contraseña

1. Adicionar el **ChangePasswordRequest**:

public class ChangePasswordRequest

{

[Required]

[StringLength(20, MinimumLength = 6)]

public string OldPassword { get; set; }

[Required]

[StringLength(20, MinimumLength = 6)]

public string NewPassword { get; set; }

}

1. Adicionar este método al API **AccountController**:

[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]

[HttpPost]

[Route("ChangePassword")]

public async Task<IActionResult> ChangePassword([FromBody] ChangePasswordRequest request)

{

if (!ModelState.IsValid)

{

return BadRequest(new Response

{

IsSuccess = false,

Message = "Bad request",

Result = ModelState

});

}

string email = User.Claims.FirstOrDefault(c => c.Type == ClaimTypes.NameIdentifier).Value;

User user = await \_userHelper.GetUserAsync(email);

if (user == null)

{

return NotFound("Error001");

}

IdentityResult result = await \_userHelper.ChangePasswordAsync(user, request.OldPassword, request.NewPassword);

if (!result.Succeeded)

{

var message = result.Errors.FirstOrDefault().Description;

return BadRequest(new Response

{

IsSuccess = false,

Message = "Error005"

});

}

return Ok(new Response { IsSuccess = true });

}

1. Probamos en Postman.
2. En el MVC **AccountController** renombrar **ChangePassword** por **ChangePasswordMVC** para evitar que entre en conflicto con el controlador API.
3. Publicamos de nuevo en Azure.

# API crear órdenes

1. Crear la enumeración **PaymentMethod**:

public enum PaymentMethod

{

Cash,

CreditCard

}

1. Crear la clase **OrderDetailResponse**:

public class OrderDetailResponse

{

public int Id { get; set; }

public ProductResponse Product { get; set; }

public float Quantity { get; set; }

public string Remarks { get; set; }

public decimal? Value => (decimal)Quantity \* Product?.Price;

}

1. Crear la clase **OrderResponse**:

public class OrderResponse

{

public int Id { get; set; }

public DateTime Date { get; set; }

public DateTime DateLocal => Date.ToLocalTime();

public UserResponse User { get; set; }

public OrderStatus OrderStatus { get; set; }

public DateTime? DateSent { get; set; }

public DateTime? DateSentLocal => DateSent?.ToLocalTime();

public DateTime? DateConfirmed { get; set; }

public DateTime? DateConfirmedLocal => DateSent?.ToLocalTime();

public string Remarks { get; set; }

public PaymentMethod PaymentMethod { get; set; }

public ICollection<OrderDetailResponse> OrderDetails { get; set; }

public int Lines => OrderDetails == null ? 0 : OrderDetails.Count;

public float Quantity => OrderDetails == null ? 0 : OrderDetails.Sum(od => od.Quantity);

public decimal Value => OrderDetails == null ? 0 : OrderDetails.Sum(od => od.Value).Value;

}

1. Modificamos la entidad **Order**:

public class Order

{

public int Id { get; set; }

public DateTime Date { get; set; }

[DisplayFormat(DataFormatString = "{0:yyyy/MM/dd hh:mm}")]

[Display(Name = "Date")]

public DateTime DateLocal => Date.ToLocalTime();

public User User { get; set; }

public OrderStatus OrderStatus { get; set; }

[DisplayFormat(DataFormatString = "{0:yyyy/MM/dd hh:mm}")]

[Display(Name = "Date Sent")]

public DateTime? DateSent { get; set; }

[DisplayFormat(DataFormatString = "{0:yyyy/MM/dd hh:mm}")]

[Display(Name = "Date Sent")]

public DateTime? DateSentLocal => DateSent?.ToLocalTime();

[DisplayFormat(DataFormatString = "{0:yyyy/MM/dd hh:mm}")]

[Display(Name = "Date Confirmed")]

public DateTime? DateConfirmed { get; set; }

[DisplayFormat(DataFormatString = "{0:yyyy/MM/dd hh:mm}")]

[Display(Name = "Date Confirmed")]

public DateTime? DateConfirmedLocal => DateSent?.ToLocalTime();

[DataType(DataType.MultilineText)]

public string Remarks { get; set; }

[Display(Name = "Payment Method")]

public PaymentMethod PaymentMethod { get; set; }

public ICollection<OrderDetail> OrderDetails { get; set; }

public int Lines => OrderDetails == null ? 0 : OrderDetails.Count;

public float Quantity => OrderDetails == null ? 0 : OrderDetails.Sum(od => od.Quantity);

public decimal Value => OrderDetails == null ? 0 : OrderDetails.Sum(od => od.Value);

}

1. Correr los comandos para actualizar la BD.
2. Crear el API **OrdersController**:

[ApiController]

[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]

[Route("api/[controller]")]

public class OrdersController : ControllerBase

{

private readonly DataContext \_context;

private readonly IUserHelper \_userHelper;

public OrdersController(DataContext context, IUserHelper userHelper)

{

\_context = context;

\_userHelper = userHelper;

}

[HttpPost]

public async Task<IActionResult> PostOrder([FromBody] OrderResponse request)

{

if (!ModelState.IsValid)

{

return BadRequest();

}

string email = User.Claims.FirstOrDefault(c => c.Type == ClaimTypes.NameIdentifier).Value;

User user = await \_userHelper.GetUserAsync(email);

if (user == null)

{

return NotFound("Error001");

}

Order order = new Order

{

Date = DateTime.UtcNow,

OrderDetails = new List<OrderDetail>(),

OrderStatus = OrderStatus.Pending,

PaymentMethod = request.PaymentMethod,

Remarks = request.Remarks,

User = user

};

foreach (OrderDetailResponse item in request.OrderDetails)

{

Product product = await \_context.Products.FindAsync(item.Product.Id);

if (product == null)

{

return NotFound("Error002");

}

order.OrderDetails.Add(new OrderDetail

{

Price = product.Price,

Product = product,

Quantity = item.Quantity,

Remarks = item.Remarks

});

}

\_context.Orders.Add(order);

await \_context.SaveChangesAsync();

return Ok(order);

}

}

1. Cómo es complejo armar el request, queda pendiente probarlo en Postman.
2. Borrar la BD en Azure y publicar nuevamente.

# Mejorar la forma como se ven los comentarios

Gracias a **Jimmy Dávila** que me compartió este código:

1. Adicionar este estado a **OrderStatus**:

<div class="col-md-7">

<div class="panel panel-default">

<div class="panel-heading">

<h3 class="panel-title">Qualifications</h3>

</div>

<div class="panel-body">

<table class="table table-hover table-responsive table-striped" id="MyTableQualifications">

<colgroup>

<col span="1" />

<col span="1" />

<col span="1" />

<col span="1" width="400" />

</colgroup>

<thead>

<tr>

<th>

@Html.DisplayNameFor(model => model.Qualifications.FirstOrDefault().DateLocal)

</th>

<th>

@Html.DisplayNameFor(model => model.Qualifications.FirstOrDefault().User.Email)

</th>

<th>

@Html.DisplayNameFor(model => model.Qualifications.FirstOrDefault().Score)

</th>

<th>

@Html.DisplayNameFor(model => model.Qualifications.FirstOrDefault().Remarks)

</th>

</tr>

</thead>

<tbody>

@foreach (var item in Model.Qualifications)

{

<tr>

<td>

@Html.DisplayFor(modelItem => item.DateLocal)

</td>

<td>

@Html.DisplayFor(modelItem => item.User.Email)

</td>

<td>

@Html.DisplayFor(modelItem => item.Score)

</td>

<td>

@{

string[] noteWords = item.Remarks.Split(' ');

if (noteWords.Count() > 7)

{

<a class="" role="button" data-toggle="collapse" href="#collapseSum@(item.Id)" aria-expanded="false" aria-controls="collapseSum@(item.Id)">

@(string.Join(" ", noteWords.Take(7)) + "...")

</a>

<div class="collapse" id="collapseSum@(item.Id)">

<div class="well">

@(string.Join(" ", noteWords.Skip(7)))

</div>

</div>

}

else

{

@item.Remarks

}

}

</td>

</tr>

}

</tbody>

</table>

</div>

</div>

</div>

1. Probamos.

# Administrar pedidos

1. Modificar la vista **Details** de **ProductsController**:

public enum OrderStatus

{

Pending,

Spreading,

Sent,

Confirmed,

Cancelled

}

1. Adicionar estas anotaciones a **Order**:

[Display(Name = "Order Status")]

public OrderStatus OrderStatus { get; set; }

...

[DisplayFormat(DataFormatString = "{0:N0}")]

public int Lines => OrderDetails == null ? 0 : OrderDetails.Count;

[DisplayFormat(DataFormatString = "{0:N2}")]

public float Quantity => OrderDetails == null ? 0 : OrderDetails.Sum(od => od.Quantity);

[DisplayFormat(DataFormatString = "{0:C2}")]

public decimal Value => OrderDetails == null ? 0 : OrderDetails.Sum(od => od.Value);

1. Crear el **OrdersController**:

[Authorize(Roles = "Admin")]

public class OrdersController : Controller

{

private readonly DataContext \_context;

public OrdersController(DataContext context)

{

\_context = context;

}

public async Task<IActionResult> Index()

{

return View(await \_context.Orders

.Include(p => p.User)

.Include(p => p.OrderDetails)

.ToListAsync());

}

}

1. Adicionar la vista **Index**:

@model IEnumerable<OnSale.Web.Data.Entities.Order>

@{

ViewData["Title"] = "Index";

}

<link rel="stylesheet" href="https://cdn.datatables.net/1.10.19/css/jquery.dataTables.min.css" />

<br />

<div class="row">

<div class="col-md-12">

<div class="panel panel-default">

<div class="panel-heading">

<h3 class="panel-title">Orders</h3>

</div>

<div class="panel-body">

<table class="table table-hover table-responsive table-striped" id="MyTable">

<thead>

<tr>

<th>

@Html.DisplayNameFor(model => model.Date)

</th>

<th>

@Html.DisplayNameFor(model => model.User.FullName)

</th>

<th>

@Html.DisplayNameFor(model => model.OrderStatus)

</th>

<th>

@Html.DisplayNameFor(model => model.DateSent)

</th>

<th>

@Html.DisplayNameFor(model => model.DateConfirmed)

</th>

<th>

@Html.DisplayNameFor(model => model.PaymentMethod)

</th>

<th>

@Html.DisplayNameFor(model => model.Lines)

</th>

<th>

@Html.DisplayNameFor(model => model.Quantity)

</th>

<th>

@Html.DisplayNameFor(model => model.Value)

</th>

<th width="70px"></th>

</tr>

</thead>

<tbody>

@foreach (var item in Model)

{

<tr>

<td>

@Html.DisplayFor(modelItem => item.DateLocal)

</td>

<td>

@Html.DisplayFor(modelItem => item.User.FullName)

</td>

<td>

@Html.DisplayFor(modelItem => item.OrderStatus)

</td>

<td>

@Html.DisplayFor(modelItem => item.DateSentLocal)

</td>

<td>

@Html.DisplayFor(modelItem => item.DateConfirmedLocal)

</td>

<td>

@Html.DisplayFor(modelItem => item.PaymentMethod)

</td>

<td>

@Html.DisplayFor(modelItem => item.Lines)

</td>

<td>

@Html.DisplayFor(modelItem => item.Quantity)

</td>

<td>

@Html.DisplayFor(modelItem => item.Value)

</td>

<td>

<a asp-action="Edit" asp-route-id="@item.Id" class="btn btn-warning"><i class="glyphicon glyphicon-pencil"></i></a>

<a asp-action="Details" asp-route-id="@item.Id" class="btn btn-info"><i class="glyphicon glyphicon-align-justify"></i></a>

</td>

</tr>

}

</tbody>

</table>

</div>

</div>

</div>

</div>

@section Scripts {

@{await Html.RenderPartialAsync("\_ValidationScriptsPartial");}

<script src="//cdn.datatables.net/1.10.19/js/jquery.dataTables.min.js"></script>

<script type="text/javascript">

$(document).ready(function () {

$('#MyTable').DataTable();

});

</script>

}

1. Agregar el llamado en el menú:

@if (User.Identity.IsAuthenticated && User.IsInRole("Admin"))

{

<li><a asp-area="" asp-controller="Countries" asp-action="Index">Countries</a></li>

<li><a asp-area="" asp-controller="Categories" asp-action="Index">Categories</a></li>

<li><a asp-area="" asp-controller="Orders" asp-action="Index">Orders</a></li>

<li><a asp-area="" asp-controller="Products" asp-action="Index">Products</a></li>

<li><a asp-area="" asp-controller="Account" asp-action="Index">Users</a></li>

}

1. Probemos lo que llevamos hasta el momento.
2. Adicionar esta anotación a **Country**:

[MaxLength(50)]

[Required]

[Display(Name = "Country")]

public string Name { get; set; }

1. Adicionar esta anotación a **Department**:

[MaxLength(50)]

[Required]

[Display(Name = "Department")]

public string Name { get; set; }

1. Adicionar esta anotación a **City**:

[MaxLength(50)]

[Required]

[Display(Name = "City")]

public string Name { get; set; }

1. Cambiar el formato **"{0:yyyy/MM/dd hh:mm}"** por **"{0:yyyy/MM/dd hh:mm tt}"** en todo el proyecto.
2. Adicionar estas anotaciones a **OrderDetail**:

[DisplayFormat(DataFormatString = "{0:N2}")]

public float Quantity { get; set; }

[DisplayFormat(DataFormatString = "{0:C2}")]

public decimal Price { get; set; }

[DataType(DataType.MultilineText)]

public string Remarks { get; set; }

[DisplayFormat(DataFormatString = "{0:C2}")]

public decimal Value => (decimal)Quantity \* Price;

1. Adicionar este método al **OrdersController**:

public async Task<IActionResult> Details(int? id)

{

if (id == null)

{

return NotFound();

}

Order order = await \_context.Orders

.Include(o => o.User)

.ThenInclude(u => u.City)

.Include(o => o.OrderDetails)

.ThenInclude(od => od.Product)

.ThenInclude(od => od.Category)

.Include(o => o.OrderDetails)

.ThenInclude(od => od.Product)

.ThenInclude(od => od.ProductImages)

.FirstOrDefaultAsync(o => o.Id == id);

if (order == null)

{

return NotFound();

}

return View(order);

}

1. Adicionar la vista **Details**:

@model OnSale.Web.Data.Entities.Order

@{

ViewData["Title"] = "Details";

}

<link rel="stylesheet" href="https://cdn.datatables.net/1.10.19/css/jquery.dataTables.min.css" />

<h2>Details</h2>

<h4>Order</h4>

<hr />

<div class="row">

<div class="col-md-4">

<dl class="dl-horizontal">

<dt>

@Html.DisplayNameFor(model => model.Date)

</dt>

<dd>

@Html.DisplayFor(model => model.DateLocal)

</dd>

<dt>

@Html.DisplayNameFor(model => model.User.Document)

</dt>

<dd>

@Html.DisplayFor(model => model.User.Document)

</dd>

<dt>

@Html.DisplayNameFor(model => model.User.FullName)

</dt>

<dd>

@Html.DisplayFor(model => model.User.FullName)

</dd>

<dt>

@Html.DisplayNameFor(model => model.User.PhoneNumber)

</dt>

<dd>

@Html.DisplayFor(model => model.User.PhoneNumber)

</dd>

<dt>

@Html.DisplayNameFor(model => model.User.City.Name)

</dt>

<dd>

@Html.DisplayFor(model => model.User.City.Name)

</dd>

<dt>

@Html.DisplayNameFor(model => model.User.Email)

</dt>

<dd>

@Html.DisplayFor(model => model.User.Email)

</dd>

<dt>

@Html.DisplayNameFor(model => model.OrderStatus)

</dt>

<dd>

@Html.DisplayFor(model => model.OrderStatus)

</dd>

</dl>

</div>

<div class="col-md-6">

<dl class="dl-horizontal">

<dt>

@Html.DisplayNameFor(model => model.DateSent)

</dt>

<dd>

@Html.DisplayFor(model => model.DateSentLocal)

</dd>

<dt>

@Html.DisplayNameFor(model => model.DateConfirmed)

</dt>

<dd>

@Html.DisplayFor(model => model.DateConfirmedLocal)

</dd>

<dt>

@Html.DisplayNameFor(model => model.PaymentMethod)

</dt>

<dd>

@Html.DisplayFor(model => model.PaymentMethod)

</dd>

<dt>

@Html.DisplayNameFor(model => model.Lines)

</dt>

<dd>

@Html.DisplayFor(model => model.Lines)

</dd>

<dt>

@Html.DisplayNameFor(model => model.Quantity)

</dt>

<dd>

@Html.DisplayFor(model => model.Quantity)

</dd>

<dt>

@Html.DisplayNameFor(model => model.Value)

</dt>

<dd>

@Html.DisplayFor(model => model.Value)

</dd>

<dt>

@Html.DisplayNameFor(model => model.Remarks)

</dt>

<dd>

@Html.DisplayFor(model => model.Remarks)

</dd>

</dl>

</div>

</div>

<div>

<a asp-action="Edit" asp-route-id="@Model.Id" class="btn btn-warning">Change Status</a>

<a asp-action="Index" class="btn btn-success">Back to List</a>

</div>

<br />

<div class="row">

<div class="col-md-12">

<div class="panel panel-default">

<div class="panel-heading">

<h3 class="panel-title">Details</h3>

</div>

<div class="panel-body">

<table class="table table-hover table-responsive table-striped" id="MyTable">

<thead>

<tr>

<th>

@Html.DisplayNameFor(model => model.OrderDetails.FirstOrDefault().Product.Name)

</th>

<th>

@Html.DisplayNameFor(model => model.OrderDetails.FirstOrDefault().Product.ImageFullPath)

</th>

<th>

@Html.DisplayNameFor(model => model.OrderDetails.FirstOrDefault().Product.Category.Name)

</th>

<th>

@Html.DisplayNameFor(model => model.OrderDetails.FirstOrDefault().Remarks)

</th>

<th>

@Html.DisplayNameFor(model => model.OrderDetails.FirstOrDefault().Price)

</th>

<th>

@Html.DisplayNameFor(model => model.OrderDetails.FirstOrDefault().Quantity)

</th>

<th>

@Html.DisplayNameFor(model => model.OrderDetails.FirstOrDefault().Value)

</th>

</tr>

</thead>

<tbody>

@foreach (var item in Model.OrderDetails)

{

<tr>

<td>

@Html.DisplayFor(modelItem => item.Product.Name)

</td>

<td>

<img src="@item.Product.ImageFullPath" style="width:100px;height:100px;max-width: 100%; height: auto;" />

</td>

<td>

@Html.DisplayFor(modelItem => item.Product.Category.Name)

</td>

<td>

@Html.DisplayFor(modelItem => item.Remarks)

</td>

<td>

@Html.DisplayFor(modelItem => item.Price)

</td>

<td>

@Html.DisplayFor(modelItem => item.Quantity)

</td>

<td>

@Html.DisplayFor(modelItem => item.Value)

</td>

</tr>

}

</tbody>

</table>

</div>

</div>

</div>

</div>

@section Scripts {

@{await Html.RenderPartialAsync("\_ValidationScriptsPartial");}

<script src="//cdn.datatables.net/1.10.19/js/jquery.dataTables.min.js"></script>

<script type="text/javascript">

$(document).ready(function () {

$('#MyTable').DataTable();

});

</script>

}

1. Adicionamos este método al **ICombosHelper**:

IEnumerable<SelectListItem> GetOrderStatuses();

1. Adicionamos este método al **CombosHelper**:

public IEnumerable<SelectListItem> GetOrderStatuses()

{

return new List<SelectListItem>

{

new SelectListItem { Value = "0", Text = OrderStatus.Pending.ToString() },

new SelectListItem { Value = "1", Text = OrderStatus.Spreading.ToString() },

new SelectListItem { Value = "2", Text = OrderStatus.Sent.ToString() },

new SelectListItem { Value = "3", Text = OrderStatus.Confirmed.ToString() },

new SelectListItem { Value = "4", Text = OrderStatus.Cancelled.ToString() }

};

}

1. Creamos la **ChangeOrderStatusViewModel**:

public class ChangeOrderStatusViewModel

{

public int Id { get; set; }

public int OrderStatusId { get; set; }

public IEnumerable<SelectListItem> OrderStatuses { get; set; }

[DataType(DataType.DateTime)]

public DateTime Date { get; set; }

}

1. Adicionar estos métodos al **OrdersController**, primero inyectamos el **ICombosHelper**:

public async Task<IActionResult> Edit(int? id)

{

if (id == null)

{

return NotFound();

}

Order order = await \_context.Orders.FindAsync(id);

if (order == null)

{

return NotFound();

}

ChangeOrderStatusViewModel model = new ChangeOrderStatusViewModel

{

Date = DateTime.Now,

Id = order.Id,

OrderStatuses = \_combosHelper.GetOrderStatuses(),

OrderStatusId = (int)order.OrderStatus

};

return View(model);

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Edit(ChangeOrderStatusViewModel model)

{

if (ModelState.IsValid)

{

Order order = await \_context.Orders.FindAsync(model.Id);

order.OrderStatus = ToOrderStatus(model.OrderStatusId);

if (model.OrderStatusId == 2) // sent

{

order.DateSent = model.Date.ToUniversalTime();

}

else if(model.OrderStatusId == 3) // confirmed

{

order.DateConfirmed = model.Date.ToUniversalTime();

}

\_context.Update(order);

await \_context.SaveChangesAsync();

return RedirectToAction($"{nameof(Details)}/{model.Id}");

}

model.OrderStatuses = \_combosHelper.GetOrderStatuses();

return View(model);

}

private OrderStatus ToOrderStatus(int orderStatusId)

{

switch (orderStatusId)

{

case 0: return OrderStatus.Pending;

case 1: return OrderStatus.Spreading;

case 2: return OrderStatus.Sent;

case 3: return OrderStatus.Confirmed;

default: return OrderStatus.Cancelled;

}

}

1. Adicionar la vista **Edit**:

@model OnSale.Web.Models.ChangeOrderStatusViewModel

@{

ViewData["Title"] = "Edit";

}

<h2>Change</h2>

<h4>Order Status</h4>

<hr />

<div class="row">

<div class="col-md-4">

<form asp-action="Edit">

<div asp-validation-summary="ModelOnly" class="text-danger"></div>

<input type="hidden" asp-for="Id" />

<div class="form-group">

<label asp-for="OrderStatusId" class="control-label"></label>

<select asp-for="OrderStatusId" asp-items="Model.OrderStatuses" class="form-control"></select>

<span asp-validation-for="OrderStatusId" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="Date" class="control-label"></label>

<input asp-for="Date" type="datetime-local" class="form-control" />

<span asp-validation-for="Date" class="text-danger"></span>

</div>

<div class="form-group">

<input type="submit" value="Save" class="btn btn-primary" />

<a asp-action="Details" asp-route-id="@Model.Id" class="btn btn-success">Back to Order</a>

</div>

</form>

</div>

</div>

@section Scripts {

@{await Html.RenderPartialAsync("\_ValidationScriptsPartial");}

}

1. Probamos.

# API para obtener historia de órdenes

1. Adicionar este método al API **OrdersController**:

[HttpGet]

public async Task<IActionResult> GetOrders()

{

string email = User.Claims.FirstOrDefault(c => c.Type == ClaimTypes.NameIdentifier).Value;

User user = await \_userHelper.GetUserAsync(email);

if (user == null)

{

return NotFound("Error001");

}

List<Order> orders = await \_context.Orders

.Include(o => o.User)

.ThenInclude(u => u.City)

.Include(o => o.OrderDetails)

.ThenInclude(od => od.Product)

.ThenInclude(od => od.Category)

.Include(o => o.OrderDetails)

.ThenInclude(od => od.Product)

.ThenInclude(od => od.ProductImages)

.Where(o => o.User.Id == user.Id)

.OrderByDescending(o => o.Date)

.ToListAsync();

return Ok(orders);

}

1. Probamos.
2. Publicamos de nuevo en Azure.

# API para modificar orden (cancelar orden)

1. Adicionar este método al API **OrdersController**:

[HttpPut]

public async Task<IActionResult> PutOrders([FromBody] Order order)

{

if (!ModelState.IsValid)

{

return BadRequest(ModelState);

}

string email = User.Claims.FirstOrDefault(c => c.Type == ClaimTypes.NameIdentifier).Value;

User user = await \_userHelper.GetUserAsync(email);

if (user == null)

{

return NotFound("Error001");

}

Order currentOrder = await \_context.Orders

.Include(o => o.User)

.ThenInclude(u => u.City)

.Include(o => o.OrderDetails)

.ThenInclude(od => od.Product)

.ThenInclude(od => od.Category)

.Include(o => o.OrderDetails)

.ThenInclude(od => od.Product)

.ThenInclude(od => od.ProductImages)

.FirstOrDefaultAsync(o => o.Id == order.Id && o.User.Id == user.Id);

if (currentOrder == null)

{

return NotFound();

}

currentOrder.OrderStatus = order.OrderStatus;

currentOrder.Remarks = order.Remarks;

\_context.Orders.Update(currentOrder);

await \_context.SaveChangesAsync();

return Ok(currentOrder);

}

1. Probamos.
2. Volvemos a publicar en Azure.

# Fin